Speed-up run-time reconfiguration implementation on FPGAs

Florent Berthelot, Dominique Houzet, Fabienne Nouvel

Speed-up run-time reconfiguration implementation on FPGAs

Florent Berthelot[1], Dominique Houzet[2], Fabienne Nouvel[1]
[1]CNRS UMR 6164 IETR/INSA Rennes, 20 av des Buttes de Coesmes, 35043 Rennes, France
[2]GIPSA-lab INPG, 46 avenue Flix Viallet, 38031 Grenoble cedex, France
florent.berthelot@ens.insa-rennes.fr

***Abstract*— Reconfigurable computing is certainly one of the most important emerging research topics over the last few years, in the field of digital processing architectures. The introduction of run-time reconfiguration (RTR) on FPGAs requires appropriate design flows and methodologies to fully exploit this new functionality. For that purpose we present an automatic design generation methodology for heterogeneous architectures based on Network on Chip (NoC) and FPGAs that eases and speed-up RTR implementation. We focus on how to take into account specificities of partially reconfigurable components during the design generation steps. This method automatically generates designs for both fixed and partially reconfigurable parts of a FPGA with automatic management of the reconfiguration process. Furthermore this automatic design generation enables reconfiguration pre-fetching techniques to minimize reconfiguration latency and buffer merging techniques to minimize memory requirements of the generated design. This concept has been applied to different wireless access schemes, based on a combination of OFDM and CDMA techniques. The implementation example illustrates the benefits of the proposed design methodology.**

***Keywords*: System-level design flow, Reconfigurable computing, Run-time reconfigurable FPGA-based system.**

I. Introduction

Applications such as multimedia, encryption or wireless communication require highly repetitive parallel computations and lead to incorporate hardware components into the designs to meet stringent performance and power requirements. On the other hand architecture flexibility is the key point for developing new multi-standard and multi-application systems. Application-specific integrated circuits (ASICs) are a partial solution for these needs. They can reach high performance, computation density or power efficiency but to the detriment of architecture flexibility as the computation structure is fixed. Furthermore, the non-recurring engineering costs (NREs) of ASICs have been increasing dramatically and made them not feasible or desirable for all applications especially in case of bug-fixes, updates and functionality evolutions. Flexibility is the processor's architecture paradigm. The algorithm pattern is computed temporally and sequentially in the time by few execution units from a program instruction stream. This programmability potentially occurs at each clock cycle and is applied to a general computation structure with a limited parallelism computation capacity. The data-path can be programmed to store data towards fixed memories or register elements, but can not be truly reconfigured. This kind of architecture suffers from the memory controller bottleneck and power efficiency. These architectures have a very coarse-grained reconfiguration, reported as system level. Reconfigurable architectures can provide an efficient platform for satisfying the performance, flexibility and power requirements of many embedded systems. These kinds of architectures are characterized by some specific features. They are based on a spatial computation scheme with high parallelism capacity distributed over the chip. Control of operator behaviour is distributed instead of being centralized by a global program memory. Multiple reconfigurable architectures have been developed [1] [2], they can be classified by their reconfiguration granularity. FPGAs have a logic level of reconfiguration. Communication networks and functional units are bit-level configurable. Their highly flexible structure allows to implement almost any application. A volatile configuration memory allows to configure the data-path and the array of configurable logic blocks.

Architecture granularity elevation allows either specialization and performance or the improvement of architecture flexibility as for the processors. The introduction of dynamically reconfigurable systems (DRS) has opened a new dimension of using chip area. Recently run-time reconfiguration (RTR) of FPGA parts has led to the concept of virtual hardware [3]. RTR allows more sections of an application to be mapped into hardware through a hardware updating process. A larger part of an application can be accelerated in hardware in contrast to a software computation. Partial reconfiguration ability of recent FPGAs allows updating only a specified part of the chip while the other areas remain operational and unaffected by the reconfiguration. These systems have the potential to provide hardware with flexibility similar to that of software, while leading to better performances.

However, cutting-edge applications require heterogeneous resources to efficiently deal with the large variety of signal and multimedia processing. This is achieved by mixing various processing granularities offered by general purpose processors (GPPs), digital signal processors (DSPs) and reconfigurable architectures. These processing units need to communicate through a flexible media managing heterogeneous communications. A Network on Chip (NoC) is an emerging solution. This paper presents our methodology allowing a fast integration and use of run-time reconfigurable components, namely FPGAs. This implies a convenient methodology and

conception flow which allows a fast and easy integration of run-time reconfiguration onto applications and an automatic management of reconfiguration process. Design space exploration will benefit from this improvement as operations of the application algorithm graph could be mapped either on DSP/GPP devices or statically/dynamically on FPGA devices thanks to run-time reconfiguration. The rest of the paper is organised as follows. Following the introduction, Section II gives an overview of related approaches, especially those which are focused on dealing with run-time reconfiguration from a high-level design specification. Section III then addresses the automatic design generation targeting run-time reconfigurable architectures. This design flow and methodology has been applied for the implementation of a transmitter system for future wireless networks for 4G air interface presented in section IV where two implementation examples based on a Network On Chip (NoC) and point to point communication scheme are presented. Finally section V provides a conclusion of this work and gives some indication of future work.

II. Related work

Numerous researches are focused on reconfigurable architectures and the way to exploit efficiently their potentials. Higher level of abstraction, design space exploration, hardware/software partitioning and co-design, rapid prototyping, virtual component design and integration for heterogeneous architectures; all these topics are strong trends in architectures for digital signal processing. Silva and Ferreira [5] present a hardware framework for run-time reconfiguration. The proposed architecture is based on a general-purpose CPU which is tightly connected to a dynamically reconfigurable fabric (DRF). A tool named BitLinker allows the relocation and assembly of bitstreams of individual components and is used to automate these steps which are very dependent on the underlying hardware's organisation. This approach is architecture-centred and application mapping steps on this specific platform is not addressed. This issue requires a higher level of abstraction for design specification. PaDReH [6] is a framework for the design and implementation of run-time reconfigurable systems and deals only with the DRS hardware design flow. The use of SystemC language enables a higher level abstraction for design and validation. After a translation to the RTL level a space-time scheduling and hardware partitioning is performed allowing the generation of a run-time reconfiguration controller. Bitstreams generation is based on Xilinx [7] Modular Design Flow. Craven and Athanas [8] present a high-level synthesis (HLS) framework to create HDL. The hardware/software partitioning is performed by the designer. Reconfiguration simulations and reconfiguration controller generation is allowed from a modified version of the Impulse C ANSI C-based design language supporting HLS from C to RTL HDL. Nevertheless, final implementation steps are not addressed. The EPICURE project [9] is a more comprehensive methodology framework based on an abstraction tool that estimates implementation characteristics from a C-level description of a task and a partitioning refinement tool that realizes the dynamic allocation and the scheduling of tasks according to available resources in the dynamically reconfigurable processing unit (DRPU). An intelligent interface (ICURE) between the software unit and the DRPU acts as a hardware abstraction layer and manages the reconfiguration process. Overlapping between computations and reconfigurations is not supported. Most of these above methodologies frameworks assume a model of external configuration control, mandating the use of a host processor or are tightly coupled to a specific architecture [9] [5]. Hence custom heterogeneous architectures are not supported. Few of them address methods for specifying runtime reconfiguration from a high-level down to the consideration of specific features of partially reconfigurable components for the implementation. Our proposed methodology deals with heterogeneous architectures composed of processors, FPGA or any specific circuits around a Network on Chip. The implementation of run-time reconfiguration on hardware components, especially FPGAs, is automated and eased by the use of a high-level application and architecture specification. That is handled by the SynDEx tool which allows the definition of both application and hardware from a high level and realizes an automated Hardware/software mapping and scheduling.

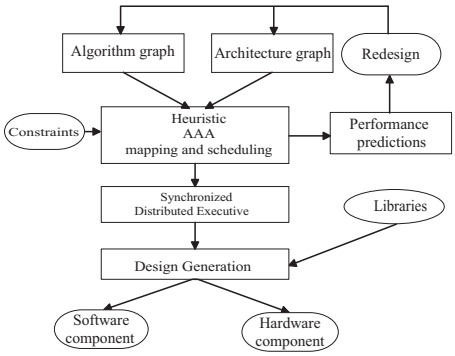


Fig. 1. SynDEx methodology flow

III. Automatic design flow targeting run-time reconfigurable architectures

Figure 1 depicts our overall methodology flow based on Syndex tool [10]. Each macrocode executive generated by Syndex is translated toward a high-level language (HDL or C/C++) for each HW or SW component. This translation produces an automatic dead-lock free code. Macro-code directives are replaced by a corresponding code from libraries (C/C++ for software components, VHDL for hardware components). Many libraries have been developed for heterogeneous platforms and we have extended SynDEx capacities to handle runtime reconfigurable components. In our methodology, the algorithm graph application description is realized at the functional level representing coarse-grained operations. This description is handled during the design flow through the use of libraries containing Ips definitions for code generation. These coarse-grained IPs are supposed to be developed to fully

exploit parallelism of their final implementation targets.

A. Generic computation structure

After selection of candidates among all operators of the algorithm graph for partial reconfiguration or parametrization, we have a set of operators that must be implemented into a run-time reconfigurable device. To achieve the design generation a generic computation structure is employed. This structure is based on buffer merging technique and functionality abstraction of operators. The aim is to obtain a single computation structure able to perform through run-time reconfiguration or parametrization the same functionalities as a static solution composed of several operators. Encapsulation of operators through a standard interface allows us to obtain a generic interface access. This encapsulation eases IP integration process with this design methodology and provides functionality abstraction to operators. This last point is helpful to easily manage reconfigurable operators with run-time reconfiguration or configuration for parametrized operators. This encapsulation is suitable for coarse-grained operators with data-flow computation. It is a conventional interface composed of an input data stream, an output data stream along with 'enable' and 'ready' signals to control computation. For the case of parametrized operators a special input is used to select the configuration of the operator. As operators can work on various data widths, the resulting operator interface has to be scaled for the worst case.

B. Design generation for run-time reconfiguration management

In order to perform reconfiguration of the dynamic part we have chosen to divide this process in two sub-parts: a Configuration manager and a Protocol builder. The 'Configuration manager' is automatically generated from our libraries according to the sequencing of operations expressed in the macro-code. A 'Configuration manager' is attached to each parametrizable operator or dynamic operator. The configuration manager is in charge of operation selection which must be executed by the configurable operator by sending configuration requests. These requests are sent only when an operation has completed its computation and if a different operation has to be executed after. So reconfigurations are performed as soon as the current operation is completed to enable configuration prefetching as described before. This functionality provides also information on the current state of the operator. This is useful to start operator computations (with signal 'enable') only when the configuration process is ended. Figure 2 shows a simple example based on two operations (j and k) which are executed successively. Labels ***M*** and ***P*** show where functionalities *'Configuration manager'* and *'Protocol builder'* respectively are implemented. Case a) shows the design generated for a non-reconfigurable component. The two operators are physically implemented and are static. Case b) is based on a parametrizable operator, the selection of configurations is managed by the configuration manager. There is no configuration latency, the operator is immediately available for computation. The signal *Config_Select* is basically a request of configuration, it results in the selection of a set of parameter among all which are internally stored. The third case c) is based on a dynamically reconfigurable operator which implements successively the two operations thanks to run-time reconfiguration. The reconfigurable part provides a virtual hardware, so at a time only one operator is physically implemented on this dynamic part. The configuration requests are sent to the protocol builder which is in charge to construct a valid reconfiguration stream in agreement with the used protocol mode (e.g selectmap).

During reconfiguration process (parametrization or partial reconfiguration) others communications or computations are allowed. Encapsulation of operators with a standard interface allows to reconfigure only the area containing the operator without altering the design around. Buffers and functionalities involved in the overall control of the dynamic area remain on a static part of the circuit. This portioning allows to reduce the size of the bitstream which must be loaded and decreases the time needed to reconfigure.

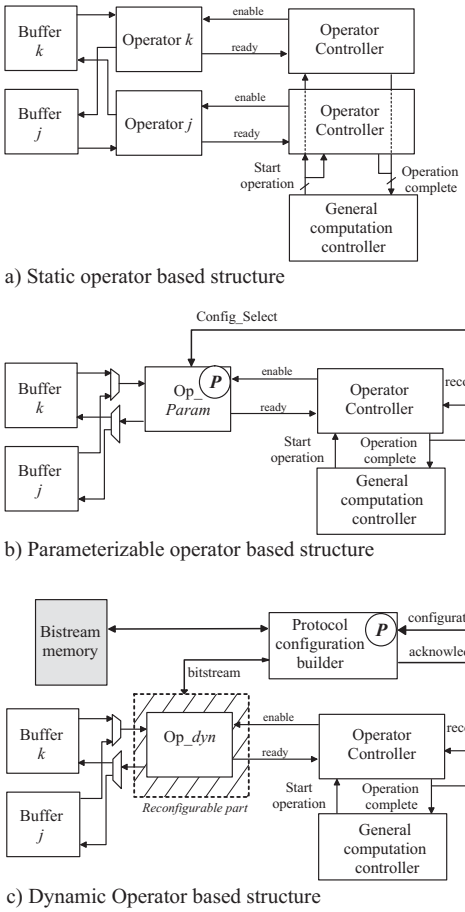


Fig. 2. Architecture comparison between a fixed/parameterized or dynamic computation based structure

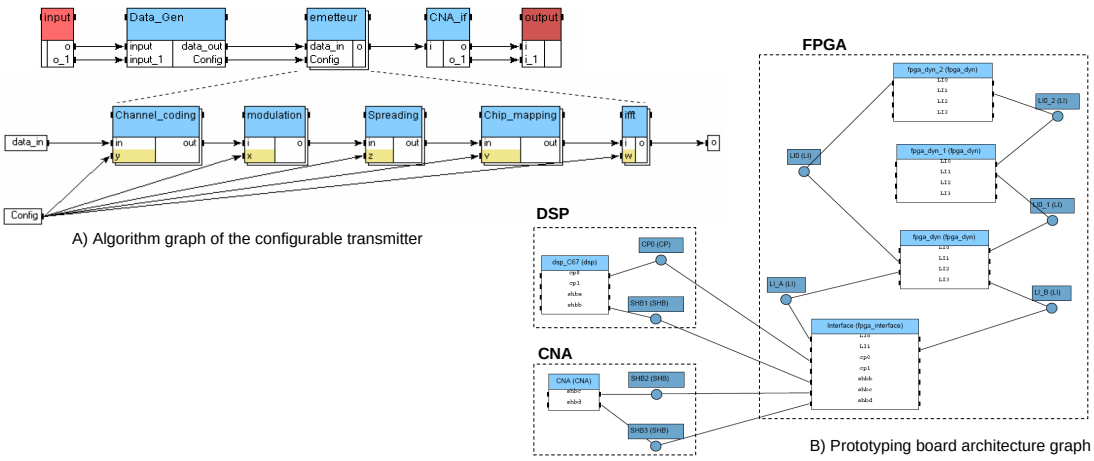


Fig. 3. Algorithm and Architecture graph of the configurable transmitter

IV. Results

Our design flow and methodology have been applied for the implementation of a transmitter for future wireless networks in a 4G-based air interface [11]. In an advanced wireless application, SDR does not just transmit. It receives informations from the channels networks, probes the propagation channel and configures the systems to achieve best performance and respond to various constraints such as bit error rate (BER) or power consumption. Hence we have considered a configurable transmitter which can switch between three transmission schemes. The basic transmission scheme is a multi-carrier modulation based on Orthogonal Frequency Division Multiplexing (OFDM). OFDM is used in many communications systems such as: ADSL, Wireless LAN 802.11, DAB/DVB or PLC. The first scheme corresponds to the most simple transmitter configuration named OFDM. The second configuration uses a Multi-Carrier Code Division Multiple Access (MC-CDMA) technique [11]. This multiple access scheme combines OFDM with spreading allowing the support of multiple users at the same time. The third configuration uses a Spread-Spectrum Multi-Carrier Multiple-Access with frequency hopping pattern (FHSS-MC-MA), as used in 802.11b (WiFi). It is a Spread Spectrum Modulation technique where signal is repeatedly switching frequencies during radio communication, minimizing probability of jamming/detection. SynDEx algorithm graph, depicted by Figure 3 A), shows the numeric computation blocks of this configurable multimode transmitter. These three transmission schemes use channel coding and perform a forward correction error (FEC), corresponding to the Channel coding block. The DSP can change the FEC to select a Reed-Solomon encoder or a convolutional encoder. Next a modulation block performs an adaptive modulation between QPSK, QAM-16 or QAM-64 modulation. For MC-CDMA and FH-SS-MC-MA schemes a Spreading block implements a Fast Hadamard Transform (FHT). This block is inactive for OFDM scheme. A chip mapping (Chip mapping block) is done in order to take into account the frequency diversity offered by OFDM modulation. This block performs either an interleaving on OFMD symbols for MC-CDMA, whereas the interleaving in FH-SS-MC-MA scheme is a frequency hopping pattern (FH) to allow user data to take advantage of the diversity in time and frequency. The OFDM modulation is performed by an Inverse Fast Fourier Transform thanks to IFFT block which also implements zero-padding process. For complexity and power consumption this IFFT can be implemented in radix-2 (IFFT-R2) or radix-4 (IFFT-R4) mode. Configuration selection (conditional entry Config) and data generation are handle by the Data Gen block, whereas CNA If block represents the interface to the CNA device of the platform.

A. Implementation on a prototyping platform

This board is composed of one DSP C6701 from Texas Instrument and one partially reconfigurable FPGA Xc2v2000 from Xilinx (10700 CLB slices). Communications between DSP and FPGA are ensured by SHB (Sundance High-speed Bus) and CP (Communication Port) communication medium from Sundance technology. We have chosen to divide the FPGA in four vertices. One is static (*Interface*) and represents pre-developed logic for FPGA interfacing. The three remaining (*FPGA_Dyn*, *FPGA_Dyn_1* and *FPGA_Dyn_2*) are run-time reconfigurable vertices. Internal communications between these parts are ensured by the LI media. Table I details the configurations and complexities of the reconfigurable transmitter computational blocks (depending on the transmission schemes). These complexities are obtained on a Xilinx VirtexII FPGA, where each slice includes two 4-input function generators (LUT). Some of these functions can be implemented thanks to available Xilinx IPs [12].

1) Functions mapping: From the characterization of computational blocks we can determine a possible implementation of the transmitter over the prototyping board. Table II summarizes this mapping. IFFT block will be implemented thanks to a parameterizable IP from Xilinx, and mapped

Transmitter configuration	Computational blocks	Channel coding		Modulation		Spreading		Chip mapping		IFFT	
• Sampling frequency = 20Mhz • Number of users = 32 • FFT = 256 points • OFDM symbol duration = 12.8 us • Frame duration = 1.32ms	**Configurations and complexities**	A B C	**Reed-Solomon encoder:** 120 slices.	A B C	**QPSK, 16QAM, 64QAM**	B C	**FHT (spreading factor: 32) :** 873 slices.	A B	**Interleaving:** 186 slices, 2 BlockRam.	A B C	**IFFT-R2 (256-points, 16 bits):** 752 slices, 3 BlockRam, 3 Mult18*18 – IP Xilinx COREGen xFFT v3.1.
		A B C	**Convolutional encoder:** 43 slices Xilinx Convolution Encoder v5.0.	A B C	**Parameterizable operator:** 60 slices			C	**Frequency Hopping Pattern (FH):** 246 slices, 2 Block-Ram.	A B C	**IFFT-R4 (256-points, 16 bits):** 1600 slices, 7 BlockRAM, 9 Mult18*18 – IP Xilinx COREGen xFFT v3.1. **Parameterizable operator:** 1600 slices - IP Xilinx COREGen xFFT v3.1.

Transmission schemes: OFDM (**A**) , MC-CDMA (**B**) , FH-SS-MC-MA (**C**)

TABLE I

CONFIGURATIONS AND COMPLEXITIES

Computation vertice	FPGA_Interface	FPGA_Dyn	FPGA_Dyn_1	FPGA_Dyn_2	DSP C67	CNA
Functional Block	Data_Gen (1) CNA_if (1)	IFFT (2) Reed Solomon Encoder(1) Convolutional Encoder (1)	Chip Mapping (3)	FHT (1) Modulation (2)	Input	Output

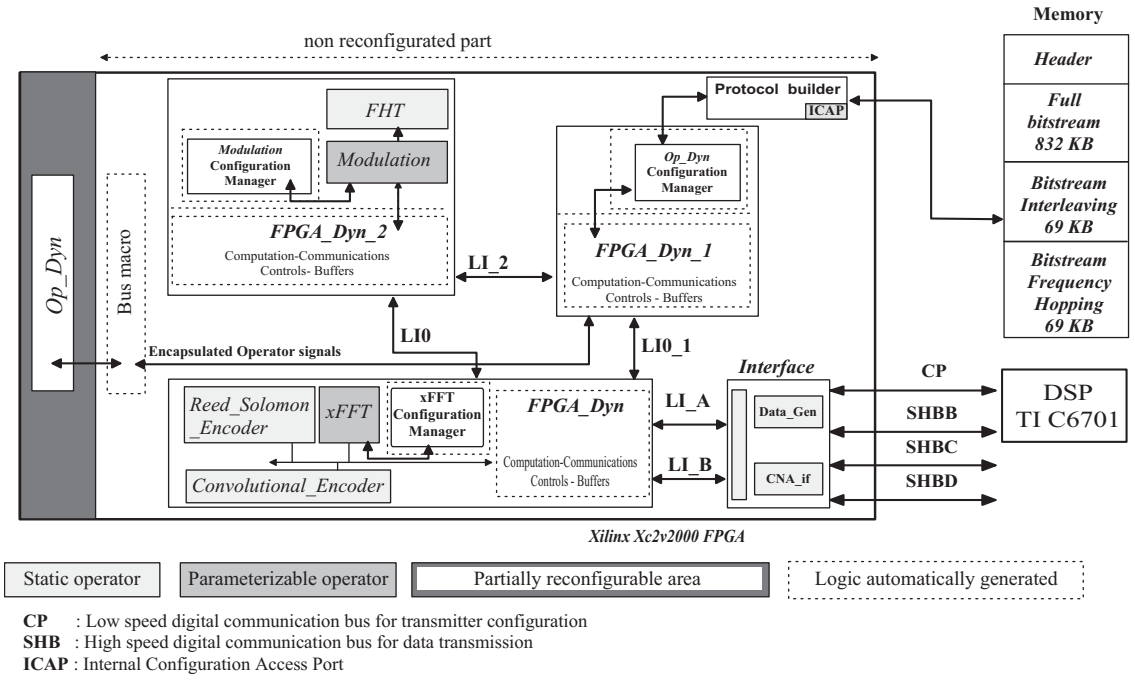
(1) : Static operator (2) : Parameterizable operator (3) : dynamic operator **Design automatically generated**

TABLE II

FUNCTIONAL BLOCKS MAPPING

on the *FPGA_Dyn* vertex. Two static operators used for the Reed-Solomon encoder and Convolutional encoder are also mapped on *FPGA_Dyn* vertex. Functionalities *Interleaving* and *FH* of the *Chip mapping* block will be sequentially implemented by a dynamic operator with run-time reconfiguration on the *FPGA_Dyn_1* vertex. Spreading block (*FHT*) will be performed through a static operator implemented on the *FPGA_dyn_2* vertex. On the same vertex the modulation is a parameterizable operator. The *Data_Gen* and *CNA_If* block are mapped on the *Interface* vertex to ensure DSP's data and configurations transmission. They will not be detailed as we focus on generation for run-time reconfigurable vertices.

2) Resulting FPGA architecture: The code, both for fixed and dynamic parts, has been automatically generated with SynDEx thanks to the libraries. However, the generation of bitstreams needs a specific flow from Xilinx called modular design [7]. Modular Design is based on a design partitioning in functional modules which are separately implemented and allocated to a specific FPGA's area. Each module is synthesized to produce a netlist and then placed and routed separately. Reconfigurable modules communicate with the other ones, both fixed and reconfigurable, through a special bus macro (3-state buffers) which is static. They guarantee that each time partial reconfiguration is performed the routing channels between modules remain unchanged. Partial bitstreams are created from the individual modules designs. The Virtex II integrates the ICAP FPGA primitive. ICAP is an acronym for Internal Configuration Access Port providing a direct access to the FPGA configuration memory and so enables a self partial reconfiguration. Figure 4 shows the resulting design of the reconfigurable transmitter which is compliant with the modular design flow for partial reconfiguration. The non reconfigurable part of the FPGA is composed of four areas resulting of the design generation of the four architecture graph vertices. Reconfigurable vertices architectures are detailed. Each are composed of the general computation/communication controller with buffers. Parametrizable and run-time reconfigurable operators have their own configuration manager. The Dynamic operator configuration manager can address reconfiguration requests to the protocol builder. The protocol builder performs partial reconfigurations thanks to the ICAP primitive and bistreams stored in external memory (Interleaving and Frequency hopping bitstreams). Only the left FPGA side is a run-time reconfigurable area and implements the dynamic operator. Parametrizable and run-time reconfigurable operators have their own configuration manager. The Dynamic operator configuration manager can address reconfiguration requests to the protocol builder. The protocol builder performs partial reconfigurations thanks to the ICAP primitive and bistreams stored in external memory (Interleaving and Frequency hopping bitstreams). Dynamic operator encapsulated signals are accessed through bus macros as circumscribed by the Modular Design flow. The size of the reconfigurable area has to be scaled to the most demanding function in logical resources, here FH function (246 slices, 2 BlockRam). Besides the shape of the reconfigurable area is constrained by the Modular Design and leads to allocating a greater area size than really necessary. In this case the area takes the full FPGA height and 12 slices width (1300 slices). This area is the only run-time reconfigured, other areas remain unchanged and are defined



CP : Low speed digital communication bus for transmitter configuration
SHB : High speed digital communication bus for data transmission
ICAP : Internal Configuration Access Port

Fig. 4. Reconfigurable transmitter architecture

once during the full FPGA configuration.

3) Numerical results of implementation: Reconfiguration operates at 50Mhz. The first and full configuration of the device takes 16 ms while the partial reconfiguration process of chip mapping functionality (operator Op Dyn) is about 2ms. That is of the order of several data frames, hence partial reconfiguration is suitable for a transmission scheme switching, as in the case of chip mapping functionality which is changed for MC-CDMA and FH-SS-MC-MA schemes. On the other hand, partial reconfiguration is too time consuming to be used for intra transmission scheme reconfiguration, it is the case if the channel coding and IFFT are implemented on the same dynamic operator. As shown in Table III, FPGA resources usage needed to implement the operator logic controls are more important with a dynamic reconfiguration implementation scheme (107+550=657 slices) as for a static and manual implementation (200 slices). The overhead is about of 450 slices to allow run-time reconfiguration of chip mapping functionality. This overhead is due to the generic VHDL structure generation, based on the macro code description. However this gap is decreasing with a greater number of configurations implemented on the dynamic operator. The aim is to take advantage of the virtual hardware ability of the architecture. However the flexibility given by this methodology and the automatic VHDL generation can overcome this hardware resource overhead. For instance we can add a Turbo convolutional encoder for the channel coding block (1133 slices, 6 BlockRam - IP Xilinx 3GPP Compliant Turbo Convolutional Codec v1.0). As the size of the reconfigurable part is fixed by the designer, any design able to be satisfied with this area constraint can be implemented.

B. Implementation based on a Network On Chip

Network on Chip (NoC) is a new concept developed since several years and intended to improve the flexibility of IP communications and the reuse of intellectual properties (IP) blocks. NoCs provide designers with a systematic, flexible and scalable framework to manage communications between a large set of IP blocks. It can also reduce IP connection wires and optimises their usage. The dynamic reconfigurability of communication paths responds to the fluctuating processing needs of embedded systems. Dataflow IPs can be connected either through point to point links or through a NoC. Tools have been proposed in order to design and customize NoCs according to their application needs. We have developed both a NoC and its corresponding tool. This NoC is one possible target of the presented methodology. This NoC is adapted and optimised to allow the plug and the management of dynamically reconfigurable IPs. Reconfigurability is one important source of flexibility when combined with a flexible communication mechanism.

1) MicroSpider NoC presentation: Our NoC [13] is built with routing nodes using a wormhole packet switching technique to carry messages. Operators are linked to the routing nodes through Network Interfaces (NI) with a FIFOlike protocol. Our NoC is customizable through an associated CAD tool [13]. Our CAD Tool is a decision and synthesis tool to help designers to obtain the had-hoc NoC depending on the application and implementation constraints. It is able to configure the various functionality of our NoC. Finally, this

Chip mapping implementation	Manual (all static)		Automatic (SynDEX) with run-time reconfiguration		
			Controls		IPs
	Controls	IPs (Interleaving+FH)	Protocol Builder	Overall dynamic part controls	Reconfigurable area capacity (used)
CLB Slices :	200	186 + 246 = 432	107	550	1300 (246)
Block RAM (18Kbits) :	2	-	-	2	14 (2)
FPGA area :	1.8 %	4 %	1 %	5.1 %	12 % (2.5 %)
Switching latency :	-	-	-	-	2 ms

TABLE III
STATIC-DYNAMIC IMPLEMENTATION COMPARISON

tool generates an optimized dedicated NoC VHDL code at RTL level.

2) Specific features: The number of ports of a router is configurable. By this way the network topology is flexible and there is no unnecessary port created. We have opted for source routing instead of a distributed one to avoid full rule table distribution and to facilitate possible run-time routing configuration. Finally, the routing technique can be simplified if the NoC topology is regular. However, it can be useful to have a routing technique that can be independent from the topology. So, a choice between two routing techniques is available. The first technique is called "dimension-ordered routing". It can be used only in the case of a regular n-dimension mesh topology. In dimension-ordered routing, each packet is routed in one dimension at a time. The second technique, called "street-sign" allows the NoC to be independent from its topology. But it imposes at source interface some tables containing paths to the destinations. Virtual channels (VC): Virtual channels are used to carry best effort (BE) and guaranteed traffic (GT). The designer can choose the number of virtual channels he wants in the NoC as well as the arbitration technique. Virtual channels make possible the separation between different traffics according to their priority despite the use of a common physical medium. In our case, priority level corresponds to its VC number. For each virtual channel, it is possible to configure independently from other virtual channels: the buffer depth, the routing technique, the arbitration technique, and the flow control technique. Network interfaces: Network interfaces are flexible and configurable to be adapted with the connected processing elements. They implement the network protocol. NIs connect IP blocks to the NoC. For NoC standardisation reasons, we made the choice of the OCP interface [14] for the communication between NI and IPs. NI uses a table to transform OCP addresses map in packet header routing information according to the NoC configuration. The network interface architecture is strongly related to SynDEx scheduling technique that requires a virtual channel controller and buffering capabilities.

3) Implementation: The operations from the application example (Figure 3 A)) are data-flow operations. Data-flow operators have FIFO like protocols. We have added specific features to our NoC in order to optimise the data-flow traffic and the plug of IPs. We have also standardized the interfaces of the NoC. A subset of OCP interface standard has been

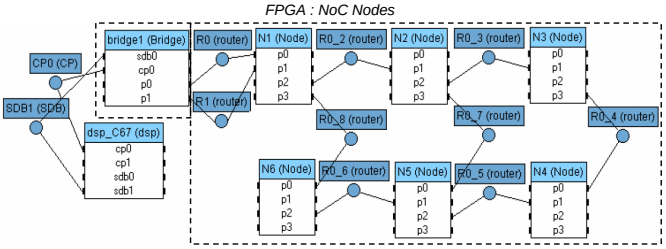


Fig. 5. Architecture graph of the NoC nodes and the DSP

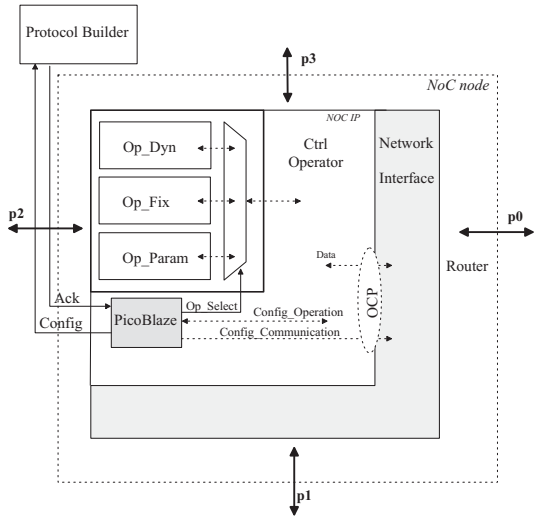


Fig. 6. NoC Node detailed view

selected and implemented. We have implemented the previous application on a six-node NoC (Figure 5) integrated in the same FPGA, with one reconfigurable area per NoC routing node, and one node dedicated to a bridge to the external C6701 DSP. Each of the five remaining nodes can be the target of any application task. Several tasks can be grouped on the same node either to be dynamically reconfigured, parametrized or fixed and simply scheduled in time. We have evaluated latency and throughput of the unloaded NoC links. These figures are introduced in SynDEx heuristic. Table IV shows the functions mapping on this NoC. As SynDEx schedules transfers in time, we use virtual channels in order to guarantee priority

of first scheduled transfers. Thus the latency is deterministic and accurate. The M4 code generated by our methodology provides all the scheduling of treatments and communications as well as the source and target of each communication. These informations are extracted and translated to a C code [15] for a Xilinx Picoblaze micro-controller in charge of the dynamic operators and parametrizable operators. Figure 6 details a NoC node structure. There is one Picoblaze for each NoC interface linked to dynamically reconfigurable operators. The Picoblaze controls the scheduling of operators, the size, the target and the starting of data transfers from the running operator. A NoC IP is the grouping of the operators, the picoblaze processor and the operators control and OCP adaptation logic. The scheduling of communications is managed with virtual channels in the NoC interfaces. They are configured by the Picoblaze. Implementation results are presented in table V. The NoC cost is similar to the point to point solution with all the advantages of flexibility and scalability. With this solution there is no need to design a dedicated architecture graph for each new application. One general purpose 2-D mesh can be selected for the architecture graph. Also, the coupling of a NoC with dynamically reconfigurable operators allow a new level of flexibility and optimisation.

N1	N2	N3	N4	N5	N6
Turbo encoder (1)	Reed Solomon encoder(3) Convolutional encoder(3)	Modulation (2)	Spreading (1)	Chip mapping (3)	IFFT (2)

TABLE IV

1: STATIC OPERATOR, 2: PARAMETERIZABLE OPERATOR, 3: DYNAMIC OPERATOR APPLICATION FUNCTION MAPPING ON THE NOC

IP	Nb Slices	Freq (MHz)	Nb BRAM
PicoBlaze	110	200	1
NoC Node	430	200	2

TABLE V

NOC NODE RESOURCES USAGE

V. CONCLUSION

We have designed and presented a flexible hardware platform for partial dynamic reconfiguration based on a heterogeneous NoC targeted to FPGA. We have integrated the dynamic reconfiguration manager directly in the NoC nodes in order to make the reconfigurable parts manageable like software tasks on a processor. We have used a methodology flow to manage automatically the partially reconfigurable parts of the FPGA. It allows to map applications over heterogeneous architectures and fully exploit advantages given by partially reconfigurable components. This design flow has the main advantage to target software components as well as hardware components to implement complex applications from a high-level functional description. This methodology is independent of the final implementation of the run-time reconfiguration which is device dependent and achieved with back-end tools. This modelling can be applied on various components of different granularities. This methodology can easily be used to introduce dynamic reconfiguration on pre-developed fixed design as well as for fast IP block integration on fixed or reconfigurable architectures. This top-down design approach makes it possible to accurately evaluate system implementation, according to functions complexity and architecture properties. Besides, the benefits of this approach fit into the SoftWare Radio requirements for efficient design methods, and adds more flexibility and adaptation capacities through partial run-time reconfiguration on FPGA-based systems.

REFERENCES

[1] R. Tessier and W. Burleson, "Reconfigurable computing and digital signal processing: A survey," *Journal of VLSI Signal Processing*, pp. pp 7–27, May/June 2001.

[2] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," *Proceedings of the conference on Design, automation and test in Europe,Munich, Germany*, pp. 642 – 649, 2001.

[3] E. L. Horta, J. W. Lockwood, D. E. Taylor, and D. Parlour, "Dynamic hardware plugins in an fpga with partial runtime reconfiguration," *Design Automation Conference (DAC)*, 2002.

[4] "Joint tactical radio system website," *http://enterprise.spawar.navy.mil*.

[5] M. L. Silva and J. C. Ferreira, "Support for partial run-time reconfiguration of platform fpgas," *Journal of Systems Architecture*, vol. 52, pp. 709–726, 2006).

[6] E. Carvalho, N. Calazans, E. Briao, and F. Moraes, "Padreh - a framework for the design and implementation of dynamically and partially reconfigurable systems," *In Proc SBCCI'04 ACM Press New York, USA*, pp. 10–15, 2004.

[7] "Xapp290: Two flows for partial reconfiguration: Module based or difference based," *http://direct.xilinx.com/bvdocs/appnotes/xapp290.pdf*.

[8] S. Craven and P. Athanas, "A high-level development framework for run-time reconfigurable applications," *Proceedings of the 9th Annual Conference on Military and Aerospace Programmable Logic Devices, MAPLD, Washington*, 2006.

[9] J. P. Diguet, G. Gogniat, J. L. Philippe, Y. L. Moullec, S. Bilavarn, C. Gamrat, K. B. Chehida, M. Auguin, X. Fornari, and P. Kajfasz, "Epicure: A partitioning and co-design framework for reconfigurable computing," *Journal of Microprocessors and Microsystems, Elsevier*, vol. Volume 30 Issue 6, pp. 367–387, September 2006.

[10] C. Sorel and Y. Lavarenne, "From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations," *Formal Methods and Models for Codesign Conference, France*, pp. 123– 132, June 2003.

[11] S. Lenours, F. Nouvel, and J.-F. Helard, "Design and implementation of mc-cdma systems for future wireless networks," *EURASIP JASP*, pp. 1604–1615, August 2004.

[12] "Xilinx intellectual property center," *http://www.xilinx.com/ipcenter/*.

[13] S. Evain, J.-P. Diguet, and D. Houzet, "A generic cad tool for efficient noc design," *Proceedings of 2004 International Symposium on Intelligent Signal Processing and Communication Systems, 2004. ISPACS 2004.*, pp. 728 – 733, 18-19 Nov. 2004.

[14] "Ocp international partnership. open core protocol specification. 2.0 release candidate," 2003.

[15] "Pccomp: Small c compiler for picoblaze," *http://www.poderico.co.uk*.